An Extension of Knowledge-Level Planning to Interval-Valued Functions

An Extension of Knowledge-Level Planning to Interval-Valued Functions

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Abstract

We investigate the problem of reasoning about numerical functions in the presence of incomplete information, sensing actions, and conditional plans. An interval-based representation is introduced into the PKS (Planning with Knowledge and Sensing) planner, as a means of compactly representing sets of possible values for numerical functions. We describe the enhancements we make to PKS, and demonstrate how such information can be used for modelling uncertain sensors and effectors. We also show how interval-valued functions can be used as a form of noisy run-time variable in plans. This paper presents a snapshot of work currently in progress.

Introduction and Motivation

An agent operating in a real-world domain often needs to do so with *incomplete information* about the state of the world. An agent with the ability to *sense* the world can also gather additional information to generate plans with *contingencies*, allowing the agent to reason about the possible outcomes of sensed information at plan time, thereby extending its ability to successfully construct plans in uncertain domains.

One particularly useful type of sensed information is *numerical* information. The ability to reason about numbers is often required in many real-world planning contexts, in order to construct plans that work with numeric state properties (e.g., the robot is 10 metres from the wall), manage limited resources (e.g., ensure the robot has enough fuel to complete the task), satisfy numeric constraints (e.g., only grasp an object if its radius is less than 10 cm), or apply arithmetic operations (e.g., advancing the robot forward one step reduces its distance from the wall by 1 metre). The importance of numerical reasoning in planning has previously been recognized with the inclusion of numeric state variables in PDDL (Fox and Long 2003), and the construction of planning systems like MetricFF (Hoffmann 2003) that are able to work with (limited forms of) numeric information.

Reasoning about numerical information under conditions of incomplete information is potentially problematic, however, especially for planners that are built on possible-world representations or sets of belief states. In such representations, the complete set of possible values for an unknown (or incompletely known) state property is often explicitly represented, e.g., by a set of states, each of which denotes a possible configuration of the actual world state. If the value of a *numeric function* is unknown, then the belief state must contain a state representing every possible mapping of the function, which could be a potentially large (or even infinite) set. Even when the range of possible values is relatively small, the number of required states can quickly grow. E.g., if a numeric function f could potentially map to any natural number between 1 and 100, then we require 100 states to capture the set of possible mappings using a possible world/belief state approach. The state explosion resulting from large sets of mappings can be computationally difficult for planners that must reason with individual states and progress (or regress) those states to construct plans.

The general problem of reasoning about knowledge and action, while avoiding the drawbacks of possible worlds, has previously been studied in formal representation languages like the situation calculus (see, e.g., (Demolombe and Pozos Parra 2000; Soutchanski 2001; Liu and Levesque 2005; Petrick 2006; Vassos and Levesque 2007)). Many of these accounts model certain (restricted) types of knowledge directly, rather than indirectly inferring such information from sets of worlds, thereby trading representational expressiveness for more tractable reasoning. (For instance, simple relational facts can be explicitly modelled by sets of predicates known to be true and sets of predicates known to be false.) One representation for modelling uncertain numerical information without possible worlds uses the notion of an *interval-valued function* as a means of capturing a set of disjunctive alternatives (Funge 1998). The idea is simple: rather than representing each possible function mapping individually across a set of worlds, a single function mapping is used and only the endpoints of the range of possible values are represented. Thus, a function f that maps from 1 to 100 can be denoted in an interval-valued form, $f = \langle 1, 100 \rangle$.

Interval-valued models of numeric information have been investigated in the planning community, especially when time is represented as a resource (see, e.g., (Edelkamp 2002; Frank and Jónsson 2003; Laborie 2003)). The idea of bounding uncertain numeric properties by intervals has also been studied in a planning context (Poggioni, Milani, and Baioletti 2003), however, to the best of our knowledge the combination of numerical reasoning with incomplete information, sensing, and contingent planning has not been fully explored. We focus on this problem in the present paper, which describes work currently in progress to extend the

PKS (Planning with Knowledge and Sensing) planner (Petrick and Bacchus 2002; 2004), by incorporating the ability to use interval-valued functions.

Our interest in adding interval-valued representations to PKS is twofold. First, PKS has always had the ability to work with simple numerical information (e.g., function (in)equalities and arithmetic operations), however, unlike other types of knowledge in PKS, its ability to reason about uncertain numerical values is limited. We believe interval-valued functions provide a compact representation that can be used to model the effects of noisy actions and knowledge, and augment PKS's existing ability to work with incomplete knowledge without using possible worlds or belief states. Second, we are also interested in tracking the results of certain types of sensed information through subsequent physical actions. For instance, if the location of a robot is sensed and the robot then moves 2 steps forward, we should be able to use such information in a planning context, even if the actual value of the robot position isn't explicitly known. We believe an interval-based representation will also be useful in modelling such situations. We explore both of these ideas in this paper.

The rest of the paper is organized as follows. In the next section we briefly review the PKS planner. We then describe a simple model of interval-valued functions. Using this model we characterize the changes we have currently implemented in PKS. We then demonstrate the extended version of PKS with a series of detailed examples. Finally, we discuss some open problems and future work, and conclude.

Planning with Knowledge and Sensing (PKS)

In this work, we aim to extend PKS (Planning with Knowledge and Sensing), a contingent planner that constructs plans in the presence of incomplete information and sensing actions (Petrick and Bacchus 2002; 2004). PKS works at the "knowledge-level" by reasoning about how the planner's knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order language, and a limited amount of inference in that subset, allowing it to support a rich representation with non-propositional features such as functions and variables. This approach differs from planners that work with propositional representations over which complete reasoning is feasible, or approaches that model incomplete knowledge based on sets of possible worlds. By working at the knowledge level, PKS can often abstract its reasoning from irrelevant distinctions that occur at the world level.

PKS is based on a generalization of STRIPS (Fikes and Nilsson 1971). In STRIPS, the state of the world is modelled by a single database. Actions update this database and, by doing so, update the planner's world model. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge. The contents of these databases have a fixed, formal interpretation in a modal logic of knowledge. Actions can modify any of the databases, which has the effect of updating the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent in each database. We briefly discuss each database below.

$\mathbf{K_f}$: This database is similar to a standard STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f is used for modelling the effects of actions that change the world. K_f can include any ground literal ℓ, where $\ell \in K_f$ means "the planner knows ℓ." K_f can also contain information about known function (in)equality mappings.

$\mathbf{K_w}$: This database models the plan-time effects of "binary" sensing actions. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at execution time this disjunction will be resolved. In such cases we will also say that the planner "knows whether ϕ." Know-whether knowledge is important since PKS can use such information to construct conditional plans with branches (see below).

$\mathbf{K_v}$: This database stores information about function values that will become known at execution time. In particular, K_v can model the plan-time effects of sensing actions that return constants, such as numeric values. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time the planner will have definite information about f's value. As a result, PKS is able to use K_v terms as "run-time variables" (Etzioni et al. 1992) or placeholders in its plans.

$\mathbf{K_x}$: This database models the planner's "exclusive-or" knowledge of literals, namely that the planner knows "exactly one of a set of literals is true." Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a particular type of disjunctive knowledge that is common in many planning scenarios, namely that "exactly one of the ℓ_i is true."

LCW: This database stores the planner's "local closed world" information (Etzioni, Golden, and Weld 1994), i.e., instances where the planner has complete information about the state of the world. We mention LCW here for completeness but will not focus on it in this paper.

PKS's databases can be inspected through a set of *primitive queries* that ask simple questions about the planner's knowledge state. Primitive queries have the following form: (i) Kp, is p known to be true?, (ii) $K_v t$, is the value of t known?, (iii) $K_w p$, is p known to be true or known to be false? (i.e., does the planner know-whether p?), or (iv) the negation of queries (i)–(iii). An inference algorithm evaluates primitive queries by checking the contents of the databases, taking into consideration the interaction between different types of knowledge.

An action in PKS is modelled by a set of *preconditions* that query the agent's knowledge state, and a set of *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a collection of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. For example, $\mathit{add}(K_f, \phi)$ adds ϕ to K_f, and $\mathit{del}(K_w, \phi)$ removes ϕ from K_w. Actions are permitted to have ADL-style context-dependent effects (Pednault 1989), where the secondary preconditions of an effect are described by lists of primitive

queries, and can employ a limited form of quantification ($\forall^K x$ and $\exists^K x$) that ranges over known instantiations of x.

PKS constructs plans by reasoning about actions in a simple forward-chaining manner: if the preconditions of an action are satisfied by the planner's knowledge state, then the action's effects are applied to the state to produce a new knowledge state. For actions with context-dependent effects, secondary preconditions are similarly evaluated against the knowledge state to determine if their effects should be applied. Planning then continues from the resulting state.

PKS can also add conditional branches to a plan, provided it has K_w knowledge. For instance, if $\phi \in K_w$ then PKS can construct two conditional branches in a plan: along one branch (the K^+ branch) ϕ is assumed to be known (i.e., ϕ is added to K_f), while along the other branch (the K^- branch) $\neg\phi$ is assumed to be known (i.e., $\neg\phi$ is added to K_f). Planning continues along each branch from the new knowledge states, until each branch satisfies the *goal*, also specified as a list of primitive queries.

Interval-Valued Functions and Knowledge

In this paper we will only focus on *functions* that map to numerical values (rather than general constants or terms). For instance, *robotLoc* $= 10$ might denote a function indicating that a robot is known to be 10 metres from a wall.

An *interval-valued function* is a function whose denotation is an *interval* of the form $\langle u, v\rangle$. The values u and v are called the *endpoints* of the interval, and indicate the bounds on a range of possible mappings for the function. Since we are primarily interested in reasoning about an agent's (incomplete) knowledge during planning, a mapping of the form $f = \langle u, v\rangle$ will mean that the value of f is known to be in the interval $\langle u, v\rangle$.[1] For instance, *robotLoc* $= \langle 5, 10\rangle$ might indicate that the distance to a wall is known to be between 5 and 10 metres. If a function maps to a *point interval* of the form $\langle u, u\rangle$, for some u, then the mapping is certain and known to be equal to u.

Each interval-valued function will be associated with a particular *number system* $\mathbb{X}$ that restricts the range of permissible intervals for a function. Typically, the number system will be one of the standard mathematical number systems (e.g., the reals $\mathbb{R}$, the natural numbers $\mathbb{N}$, the integers $\mathbb{Z}$, etc.), extended to include the points at infinity, ∞ and $-\infty$. Given a number system $\mathbb{X}$, a mapping $f = \langle u, v\rangle$ is permitted, provided $u, v \in \mathbb{X}$ and $u \leq v$. For every number system $\mathbb{X}$, the special interval $\langle \bot, \top\rangle$ represents the *maximal interval* for that number system. For instance, $\langle \bot, \top\rangle \stackrel{\text{def}}{=} \langle -\infty, \infty\rangle$ in $\mathbb{R}$, however in $\mathbb{B}$, the binary number system consisting of the two elements 0 and 1, $\langle \bot, \top\rangle \stackrel{\text{def}}{=} \langle 0, 1\rangle$. In terms of knowledge, a mapping of the form $f = \langle \bot, \top\rangle$ means that the agent considers every element of $\mathbb{X}$ as a possible mapping for f. In other words, the value of f is completely unknown to the agent.

[1] We will only focus on closed intervals whose endpoints are included as possible mappings (i.e., intervals of the form $[u, v]$ in standard mathematical notation). Open intervals (u, v), or partially-open intervals $(u, v]$ and $[u, v)$, are treated in a similar manner except for minor differences in the boundary cases.

For simplicity, we will assume that all interval-valued functions in this paper range over $\mathbb{N}$ unless otherwise indicated. Also, as an alternative to using the maximal interval $\langle \bot, \top\rangle$ to represent functional uncertainty, we will sometimes use PKS's ability to reason about incomplete information when a function is not listed in its knowledge bases.

Representing and Reasoning about Interval-Valued Knowledge in PKS

In this section we describe some of the changes we have made to PKS to support interval-valued functions. Since this paper presents a snapshot of work currently in progress, we will discuss many of these changes at a high level and leave many of the technical details for a future paper. In particular, we will focus on the representation of interval knowledge by considering changes to the K_f, K_v, K_w, and K_x databases. We will also briefly mention extensions to PKS's primitive query language and action representation.

K_f and knowledge of intervals Recall that the K_f database stores the planner's knowledge of facts, including functional equalities (e.g., $f = 10$) and inequalities (e.g., $g \neq 12$). In our extended representation we allow functions to map to interval values, provided the intervals only contain numeric constants. That is, a function like $f = \langle 5, 10\rangle$ is permitted, however, $g = \langle 5, x\rangle$ is not if x is a variable. Intuitively, a function of the form $f = \langle u, v\rangle \in K_f$ means that f is known to map to a value between u and v.

K_v and sensed intervals The K_v database is primarily used to represent the results of sensing actions that return functions. In particular, this database does not constrain the type of underlying function it can represent, i.e., whether it is an ordinary function mapping or an interval-valued mapping. Thus, K_v can immediately be used with interval-valued functions which are treated in the same way as ordinary functions. I.e., if $f \in K_v$, where f is interval valued, then the (interval) value of f is known at plan time.

However, we also extend our notion of K_v knowledge to allow *noisy* sensed information to be modelled. To do so, we specify an *interval schema* for the associated function, using a variable (x in our examples) to denote the actual value of the function. For instance, a function of the form:

$$f : \langle x - 1, x + 1\rangle \in K_v$$

means that the value of the function f is known, and f is in the range $x \pm 1$, for some x. In this case, we treat x as a special type of "run-time variable" (Etzioni et al. 1992) that acts as a placeholder to the actual value of the sensed function. The value of f in this case is "noisy" as it admits a range of possible values. In practice, we allow formulae of the form $f : \langle x - u, x + v\rangle$ in K_v, where u and v are numeric constants. This type of information will be particularly useful for tracking changes to numeric sensed values through the effects of certain physical actions.

K_w and numeric comparisons The K_w database is typically used to model sensing actions with binary outcomes, i.e., those that return one of two possible values. K_w is also important since information in this database can be used to

build conditional branches into a plan: when a conditional branch is inserted, one branch is added for each possible outcome of the sensed information.

With numeric functions (interval-based or not), certain types of numeric relations become useful in a planning context. In particular, the relational operators $=$, $\neq$, $>$, $<$, $\geq$, and $\leq$ often arise in many planning scenarios. In our extended version of PKS, we allow simple formulae using such operators to be explicitly represented in K_w, provided such formulae have the form f **op** c, where **op** $\in \{=, \neq, >, <, \geq, \leq\}$ and c is a numeric constant. Thus, $f > 5 \in K_w$ can be used to model a sensing action that determines whether f is greater than 5 or not.

Such extended K_w information can also be used to form conditional plans as usual in PKS. For a given K_w formula, two branches are added to a plan: along one branch the K_w formula is assumed to be true while along the other branch the formula is assumed to be false. Thus, if the formula $f > 5 \in K_w$ is used as the basis for a new branch point in a plan then $f > 5$ is assumed to be true in the K^+ branch, and $f \leq 5$ is assumed to be true in the K^- branch.

K_w branching is particularly important when combined with interval-based knowledge in K_f: any assumptions resulting from the addition of a branch must be combined with existing knowledge in the other databases, possibly refining or resolving that knowledge as necessary. Thus, if $f > 5$ is assumed to be true and $f = \langle 3, 10\rangle \in K_f$, then the K_f knowledge is updated and the interval is refined so that $f = \langle 6, 10\rangle$. Similarly, if $f \leq 5$ is assumed to be true then the K_f knowledge is updated so that $f = \langle 3, 5\rangle$. When used with interval-based knowledge that has a wide range of possible values, this process gives rise to a powerful technique that allows the planner to split intervals into smaller components and reason about individual subcases.

K_x versus interval-valued functions The notion of an interval-valued function has a close connection to the exclusive-or knowledge that can be represented in K_x: both types of representation can be used to model disjunctions of possible values where one, and only one, of the disjunctions can be true. For instance, in this view a formula of the form $(f = 3 \mid f = 4 \mid f = 5) \in K_x$ is similar to an interval-valued function of the form $f = \langle 3, 5\rangle$.

There are notable differences, however. In particular, K_x takes a very conservative view of physical actions that change the values of literals mentioned in K_x formula. In such cases, any formula containing a property changed by an action is completely removed from K_x since it's "exclusive-or" property may no longer hold. This is not the case for interval-valued functions. Instead, we would like to track the set of possible values for such functions through action. Each K_x formula is also restricted to a set of literals that must be explicitly enumerated as a finite disjunction. Interval-valued functions provide a more compact representation that permits continuous intervals over number systems such as the reals ($\mathbb{R}$), which cannot be modelled in K_x.

Interval-valued functions can also be included in K_x, however, and are treated in the same way as any other piece of K_x information. In particular, this means they are subject to the conservative update rules inherent in that database. We refer the reader to (Petrick and Bacchus 2004) for more information about K_x and its update rules.[2]

Primitive queries and intervals The underlying primitive query language used by PKS is unchanged with the addition of interval-valued functions. In particular, the existing query language already permits primitive queries that include numeric relational operators such as those we permit in the extended K_w database (e.g., $>$). However, we have also extended the inference procedure that evaluates primitive queries to reason with interval-based information. For instance, a query of the form $K(f > 3)$ only evaluates as true given an interval $f = \langle u, v\rangle \in K_f$ provided $u > 3$. Similarly, a query $K(g \neq 5)$ is true if both $5 < u$ and $5 > v$.

Actions and intervals Actions in our extended version of PKS are defined in a similar way to ordinary PKS actions, with preconditions and effects. Preconditions are still simply sets of primitive queries, as defined above. Effects permit updates to be made to interval-valued information through a set of simple arithmetic operations. (In this paper we only consider the arithmetic addition and subtraction operators.) In particular, we allow updates to have the form

$$\mathit{add}(K_f, f := f \pm d),$$

where f is an interval-based function and d is either a numeric constant or constant interval (i.e., no variables). In the case of a constant d, an existing interval $\langle u, v\rangle$ is updated to the resulting interval $\langle u \pm d, v \pm d\rangle$. If d itself is an interval, the process is somewhat more complicated and a new range must be calculated for the resulting interval. For instance, adding the interval $\langle 3, 5\rangle$ to $\langle 1, 2\rangle$ results in an interval $\langle 4, 7\rangle$. We currently focus on arithmetic operations that can be calculated in a straightforward manner and result in well-formed intervals.

One additional update is performed when interval-valued updates occur: K_v formulae that are specified using interval schema are also updated appropriately. That is, the interval corresponding to a K_v formula is updated in a similar manner to an ordinary interval-based function. For instance, if $f : \langle x - c, x + c\rangle \in K_v$, and an effect of the form $\mathit{add}(K_f, f := f + d)$ updates f, where d is a constant, then K_v is updated so that $f : \langle x - c + d, x + c + d\rangle \in K_v$.

Finally, we also allow actions to include ordinary database assertions, following the standard PKS rules for *add* and *del*. Thus, we can specify "noisy" knowledge through an update such as $\mathit{add}(K_f, f = \langle 3, 5\rangle)$ that adds $f = \langle 3, 5\rangle$ to K_f.

PKS planning with intervals Given the above changes to the PKS database representation, primitive query mechanism, and database update procedure, the underlying planning algorithm operates as in the unextended version of PKS. In particular, the plan generation process is treated as a search through the set of database states, starting from an initial state denoted by the initial set of databases. Plans are built in a forward-chaining manner by choosing an action to

[2] One of the open technical questions in this work is whether or not intervals of the form $f = \langle 3, 5\rangle$ actually belong in K_f, or whether a better intuitive definition would place such knowledge in an extended K_x database. We leave open the possibility of changing our current representation in the future.

Action	Effects
moveForward	$add(K_f, robotLoc := robotLoc - 1)$
moveBackward	$add(K_f, robotLoc := robotLoc + 1)$
atTarget	$add(K_w, robotLoc = targetLoc)$

Table 1: Action specifications for Example 1.

add to a plan, or by introducing conditional plan branches. Planning continues until the goal conditions are achieved along every branch of a plan, or no plan can be found. We refer the reader to (Petrick and Bacchus 2002) for more details on the actual plan generation process used by PKS.

Examples

To illustrate the above extensions, we present three simple examples of interval-based reasoning in PKS.

Example 1 Consider a robot whose location, *robotLoc*, is measured by its distance to a wall. The robot has two physical actions available to it: *moveForward* moves the robot one unit towards the wall, and *moveBackward* moves the robot one unit away from the wall. The robot also has a sensing action, *atTarget*, which senses whether the robot is at a target location specified by the function *targetLoc*. The definition of these actions is shown in Table 1. The robot's initial location is specified by the interval-valued function mapping $robotLoc = \langle 3, 5\rangle \in K_f$. The goal is to move the robot to the target location, denoted by the query $K(robotLoc = targetLoc)$. In this example, $targetLoc = 2 \in K_f$.

One solution generated by PKS is the conditional plan:

1 | *moveForward* ;
2 | *atTarget* ;
3 | *branch(robotLoc = targetLoc)*
4 | K^+ : **nop**.
5 | K^- : *moveForward* ;
6 | *atTarget* ;
7 | *branch(robotLoc = targetLoc)*
8 | K^+ : **nop**.
9 | K^- : *moveForward*.

In step 1, the *moveForward* action uniformly decreases the value of *robotLoc* in K_f by one unit so that $robotLoc = \langle 2, 4\rangle$. In step 2, *atTarget* senses whether $robotLoc = targetLoc$, which has the effect of adding $robotLoc = 2$ to K_w (i.e., the planner knows whether *robotLoc* is 2 or not). In step 3, a branch point is added to the plan based on this K_w formula, allowing the plan to consider the two possible outcomes of the K_w formula (which also has the effect of removing the formula from K_w). Along one branch (step 4), $robotLoc = 2$ is assumed to be true (i.e., $robotLoc = 2$ is added to K_f) and the goal is achieved. Along the other branch (step 5), $robotLoc \neq 2$ is assumed to be true (i.e., $robotLoc \neq 2$ is added to K_f). As a result, the interval mapping for *robotLoc* in K_f can be refined to remove 2 as a possible mapping, so that $robotLoc = \langle 3, 4\rangle$. The *moveForward* action then updates *robotLoc* further so that $robotLoc = \langle 2, 3\rangle$. The sensing action in step 6 again adds $robotLoc = 2$ to K_w. In step 7, another branch point is added to the plan. Along one branch (step 8), $robotLoc = 2$ is assumed to true, satisfying the goal. Along the other branch (step 9), $robotLoc \neq 2$ is assumed to be true. In this case, refining *robotLoc* results in the (definite) knowledge that $robotLoc = \langle 3, 3\rangle = 3$. A final *moveForward* action results in $robotLoc = 2$, satisfying the goal.

Action	Effects
noisyForward	$add(K_f, robotLoc := robotLoc - \langle 1, 2\rangle)$
withinTarget	$add(K_w, robotLoc \leq targetLoc)$

Table 2: Additional actions for Example 2.

Example 2 We next consider a robot with the *moveBackward* and *atTarget* actions from Example 1, but with *moveForward* replaced by a "noisy" movement action, *noisyForward*, which moves the robot forward either 1 or 2 units. Additionally, the robot also has a second sensing action, *withinTarget*, that determines whether or not the robot is within the target distance (where $targetLoc = 2 \in K_f$). The specification of these new actions is given in Table 2. In this example, the robot's initial location is specified by the interval-valued mapping $robotLoc = \langle 3, 4\rangle \in K_f$. The goal is to move the robot to the target location, i.e., $K(robotLoc = targetLoc)$.

One solution generated by PKS is the conditional plan:

1 | *noisyForward* ;
2 | *withinTarget* ;
3 | *branch(robotLoc ≤ targetLoc)*
4 | K^+ : *atTarget* ;
5 | *branch(robotLoc = targetLoc)*
6 | K^+ : **nop**.
7 | K^- : *moveBackward*.
8 | K^- : *noisyForward* ;
9 | *atTarget* ;
10 | *branch(robotLoc = targetLoc)*
11 | K^+ : **nop**.
12 | K^- : *moveBackward*.

Since forward movements may change the robot's position by either 1 unit or 2 units, the *noisyForward* action in step 1 results in an even less certain position for the robot, namely that $robotLoc = \langle 1, 3\rangle \in K_f$. However, the sensing action in step 2, together with the branch point in step 3, lets us split this interval into two sub-intervals. In step 4, we assume that $robotLoc \leq 2$ and consider the case that $robotLoc = \langle 1, 2\rangle$. The *atTarget* action, together with the branch in step 5, lets us divide this interval even further: in step 6, $robotLoc = 2$ and the goal is satisfied, while in step 7, $robotLoc = 1$ and a *moveBackward* action achieves the goal. In step 8 we consider the other sub-interval of the first branch, namely the interval resulting from $robotLoc > 2$, i.e., $robotLoc = 3 \in K_f$. In this case we have definite knowledge of the robot's location, however, the subsequent *noisyForward* action results in $robotLoc = \langle 1, 2\rangle$. The remainder of the plan in steps 9–12 is the same as in steps 4–7: the robot conditionally moves backwards in the case that *robotLoc* is determined to be 1, while the plan trivially achieves the goal if $robotLoc = 2$.

Example 3 In the final example, we consider a robot with the *moveBackward* action from Table 1, and the *noisyLocation* action from Table 3. In this case, *noisyLocation* is a noisy sensing action that either senses the actual value of the

Action	Effects
noisyLocation	$add(K_v, robotLoc : \langle x, x+1 \rangle)$

Table 3: Additional action for Example 3.

robot's location, or 1 unit more than the actual location. This is denoted by the notation $\langle x, x+1 \rangle$ in the action description, where x acts as a placeholder for the actual location, and the interval specifies the range of possible values. Initially, the location of the robot is unknown, i.e., *robotLoc* is not listed in the planner's databases. The goal is to ensure the robot has moved to or past the target location, i.e., $K(robotLoc \geq targetLoc)$, where $targetLoc = 2 \in K_f$.

Here, PKS can generate the simple 3-step plan:

1 | *noisyLocation* ;
2 | *moveBackward* ;
3 | *moveBackward*.

Step 1 of the plan adds the formula $robotLoc = \langle x, x+1 \rangle$ to K_v, indicating that the planner has (noisy) knowledge of the robot's location. In step 2, the result of *moveBackward* updates the planner's parametrized K_v knowledge. In particular, $robotLoc = \langle x+1, x+2 \rangle$, which has the effect of tracking the movement action in relation to the planner's (ungrounded) location information. In step 3, the second *moveBackward* action results in $robotLoc = \langle x+2, x+3 \rangle$. In this case, the planner can reason that $robotLoc \geq 2$ holds since *robotLoc* is a function over $\mathbb{N}$: since $x \geq 0$, it must be the case that $x + 2 \geq 2$.

Although the above examples are admittedly simple, they nevertheless demonstrate some interesting plan-time reasoning. In Example 1, we illustrate a case where the planner has uncertain knowledge about the location of a robot. Using interval-valued functions, we track the robot's location as physical actions change this information and sensing actions, together with conditional plan branches, produce more certain knowledge. We note that in the original version of PKS, we could represent the disjunctive nature of *robotLoc* (for $\mathbb{N}$ at least) using the K_x database, e.g., $(robotLoc = 3 \mid robotLoc = 4 \mid robotLoc = 5) \in K_x$. However, an action like *moveForward* would immediately invalidate this information, causing it to be removed, since it changes a function mentioned in the K_x formula.

In Example 2, we consider a simple case of an action with a noisy physical effect, represented by the action *noisyForward*, that changes *robotLoc* with an interval-based effect. Again, we demonstrate how the use of sensing actions together with conditional branching allows us to divide intervals into more manageable components and reason about individual subcases. Of course, this example also demonstrates that if we do not have the *right* sensing actions (e.g., *withinTarget*), then the ability to track certain intervals alone may not always be sufficient for useful reasoning.

Finally, in Example 3 we illustrate an interesting type of reasoning we are currently experimenting with: the ability to track sensed information through physical actions using a type of placeholder variable. In particular, *noisyLocation* is a noisy sensing action whose resulting (indefinite) knowledge is tracked through *moveBackward* actions. We note that in this example the K_v interval is not strictly necessary for finding a plan since only the left endpoint of the interval is used. (I.e., an action that adds $robotLoc : \langle x, x \rangle$ to K_v using a point interval would be sufficient.) However, the interval-based sensing action demonstrates one of the new features of our extended PKS representation.

Example 3 also demonstrates one of the drawbacks inherent in plan-time sensing with unknown quantities: for such values to be useful in a plan we often need to "ground" them in some way. In this case, we use knowledge of the underlying number system and the interval offset to make an assertion about a lower bound. However, PKS also has the ability to work with functions in an "unground" form, allowing them to be composed with other functions, or using them to produce a form of parametrized plan. One of the goals of this work is to extend PKS's representation of interval-valued functions so they can also be used in this way. One particular application where we believe this will be useful is in the automatic generation of plans with *loops* (Levesque 2005). (For instance, in the case of Example 3 we could imagine generating a parametrized plan that loops until a certain exit condition is achieved.) However, this extension is part of ongoing work.

Discussion and Future Work

Interval-valued functions provide an interesting middle ground between those representations that do not represent uncertainty about numerical values and those that reason with full possible-world models, or models based on probabilistic distributions. For a planner like PKS that works with a restricted representation to model particular types of knowledge, an interval-valued representation makes a good fit and offers another useful tool for knowledge-level planning. Moreover, such extensions have not been fully explored in the context of planning with incomplete information, sensing actions, and contingent plans, and this work offers the prospect of results that can be applied beyond PKS.

There are still some non-trivial technical problems to overcome. First, we are considering interval-based operations other than addition and subtraction. While we do not want to integrate a complete equation solver with our planner, we are focusing on those operations that are useful for planning and that can easily be "tracked" in simple, predictable ways. To help guide our work, we are investigating planning problems based on real-world robot domains requiring numeric reasoning. We are also exploring how existing approaches in the literature use intervals in other contexts (e.g., temporal reasoning).

Second, although intervals provide a compact means of representing a range of possible values, there are problems when those values are sparsely distributed. For instance, if f can map to the values 5, 7, and 10, then the interval $f = \langle 5, 10 \rangle$ suffices for representing the set of possible mappings, but also admits 6, 8, and 9 as possible values. While such intervals may not be problematic, depending on the task, they are potentially less accurate and may incur more reasoning than needed. To overcome these potential drawbacks, we are also investigating functions that map to *interval sets* consisting of a finite number of intervals. In

such cases, a function f could map to an interval set of the form $f = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n\}$, where each $\mathcal{I}_i$ is an interval. In PKS, this would combine the interval-based representation described here in a form of "extended-K_x" knowledge. To avoid situations of excessive fragmentation, with large numbers of intervals, we will initially bound the number of intervals allowed in a set, and in some cases use a single wide interval in the place of multiple intervals, if necessary.

Finally, we have not focused on the efficiency of PKS's plan generation process in this paper but have instead considered particular representation and reasoning problems. (We note that all the examples presented in this paper can be generated in less than a second using PKS on a single CPU running at 1.86 GHz with 2Gb of RAM.) In other work, we are also addressing the problem of scaling up PKS's performance by adapting heuristic search techniques to the state spaces produced by PKS. While interval-based representations may complicate this process somewhat, we believe that the compilation techniques of (Petrick 2006) could be adapted to this problem, allowing interval-based functions to be treated in a similar fashion to ordinary functions.

This paper presents a snapshot of work in progress. It also forms part of a larger research agenda aimed at transforming standard contingent planning domains (e.g., domains that can be represented in planners like Contingent-FF (Hoffmann and Brafman 2005)) into knowledge-level forms. It also builds on theoretical work in the situation calculus (Funge 1998; Demolombe and Pozos Parra 2000; Petrick 2006) that we are currently extending, with a focus on the construction of practical planning systems.

Acknowledgements

This work was partially funded by the Natural Sciences and Engineering Research Council (NSERC) scholarship program while the author was at the University of Toronto, and by the European Commission through the JAMES (Grant No. 270435) and Xperience (Grant No. 270273) projects.

References

Demolombe, R., and Pozos Parra, M. P. 2000. A simple and tractable extension of situation calculus to epistemic logic. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS-2000)*, 515–524.

Edelkamp, S. 2002. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed world reasoning with updates. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, 178–189.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Journal of Constraints, Special Issue on Constraints and Planning* 8:339–364.

Funge, J. 1998. Interval-valued epistemic fluents. In *AAAI Fall Symposium on Cognitive Robotics*, 23–25.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 71–80.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.

Levesque, H. J. 2005. Planning with loops. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, 509–515.

Liu, Y., and Levesque, H. J. 2005. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, 522–527.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, 324–332.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-04)*, 2–11.

Petrick, R. P. A. 2006. *A Knowledge-level approach for effective acting, sensing, and planning*. Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

Poggioni, V.; Milani, A.; and Baioletti, M. 2003. Managing interval resources in automated planning. *Journal of Information Theories and Applications* 10:211–218.

Soutchanski, M. 2001. A correspondence between two different solutions to the projection task with sensing. In *Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 2001). http://www.cs.nyu.edu/faculty/davise/commonsense01/.*

Vassos, S., and Levesque, H. 2007. Progression of situation calculus action theories with incomplete information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2029–2034.